# 整数の計算

### *演習問題 2.1 文法事項*

1）あなたの利用している、コンピュータや C コンパイラにおける int 型は、short 型と同じか、long 型と同じか調べなさい。

**解答例** 皆さん、ご自分のシステムで是非調べてみて下さい。調べる方法として、真面目にコンパイラのマニュアルを読むのも良いことですが、データ型のサイズを調べるプログラムを作ることもできます。このためには、本書の中では紹介しなかった **sizeof** 演算子を用います。**sizeof** 演算子は、後に続くカッコの中に書かれたデータ型や変数の大きさをバイト単位で教えてくれます。

**sizeof** 演算子を用いて、**int** 型、**short** 型、**long** 型のサイズを調べるためのプログラムを解答プログラム 2.1 に示しておきます。　実行結果を見ると、筆者のシステムでは、**int** 型 4 バイト、**short** 型 2 バイト、**long** 型 4 バイトで、**int** 型は **long** 型と同じ大きさであることが判ります。

この後の章を勉強されて、新たなデータ型に出会ったら、同じようなプログラムを作ってみて、様々なデータ型のサイズを調べてみるのも面白いと思いますよ。

**解答プログラム 2.1**　データ型のサイズを調べる

```
/************************************
    p2_1.c
    データ型のサイズを調べる
    Tadaaki Shimizu  2006.11.02
************************************/
#include <stdio.h>

main()
{
    printf("int 型のサイズは    %d バイトです。\n", sizeof(int));
    printf("short 型のサイズは  %d バイトです。\n", sizeof(short));
    printf("long 型のサイズは   %d バイトです。\n", sizeof(long));
    exit(0);
}


[実行例]
% ./p2_1
int 型のサイズは    4 バイトです。
short 型のサイズは  2 バイトです。
long 型のサイズは   4 バイトです。
%
```

2）次の定数のデータ型を答えなさい。また、文法的に間違っているものと、文法的には正しいけれども、読みやすさの点からよくないと思われる定数の表記法を a ) から f ) のうちから選び、その理由を述べなさい。

a ) `128`　　b ) `40000`　　c ) `45L`　　d ) `123l`

e ) `0173`　　f ) `0584`　　g ) `0xff7a`　　h ) `0xhea1`

**解答例**　1. 文法的に間違っているもの

f ) **`0584`**：　**`0`** で始まる定数は整数型の 8 進数表現であり、使える数字は **`0`** 〜 **`7`** までのはずなのに、**`8`** が使われている。

h) **`0xhea1`**：　**`0x`** で始まる定数は整数型の 16 進数表現であり、使える数字は **`0`** 〜 **`9`** までと **`a`** 〜 **`f`** までなのに、**`h`** が使われている。

2. 読みやすさの点からよくないもの

d)　**`123l`**：　定数が **`long`** 型であることを示す **`L`** は、小文字の **`l`** でなく大文字の **`L`** を使うことを強く勧めます。この例は、**`long`** 型の **`123`** ですが、**`123l`**( 千二百三十一 ) と読み間違えやすいでしょう。

3）C 言語による次の式の演算結果は何になりますか。

a ) `12 + 3 / 2`　b ) `- 2 * 6 - 5`　c ) `54000L * (12000L / 1000L)`

**解答例**　a) **`13`**

**`3 / 2`** が先に計算されますが、整数割り算のため小数点以下は切り捨てられて、答えは **`1`** となります。この答えと **`12`** が足されて、演算結果は **`13`** となります。

b) **`-17`**

**`2`** に単項演算子の **`-`** が作用して **`-2`** となり、これと **`6`** が掛け算されて **`-12`** となります。さらに、これから **`5`** が引かれて、演算結果は **`-17`** となります。

c) **`648000L`**

カッコのために、**`12000L / 1000L`** が先に演算されて **`12L`** となり、これと **`54000L`** が掛け算されて、演算結果は **`648000L`** となります。

4）次の変数のうち、文法的に正しくない変数名を持つ変数はどれか。また、文法的には正しくても読みやすさの点からよくないと思われる変数はどれか。理由とともにあげなさい。

a ) `data_sum`　　b ) `tel_$number`　　c ) `ab`　　d ) `1_output`

e ) `firstEntry`　　f ) `double`　　g ) `number of data`

**解答例**　1. 文法的に間違っているもの

b) **`tel_$number`**：　変数名に使える文字は、英文字と数字と「_( アンダー・スコア )」だけです。この例では、**`$`** が使われているので文法違反です。

d) **`1_output`**：　変数名は数字で始めることはできません。

f) **`double`**：　これは C 言語の予約語なので変数名として使えません。

g) **`number of data`**：　変数名にスペースを含めることはできません。

2. 読みやすさの点からよくないもの

c) **ab**：このような無意味な変数名は読みにくいプログラムの原因となります。

筆者が利用しているコンパイラ (gcc バージョン 3.3.1) では、文法違反である b) の変数名が使えます。皆さんのコンパイラでも、文法違反の変数名が使える場合があるかも知れません。しかし、たまたま動くからといって、文法違反をしていると後で困ったことになるかもしれません。あなたは、いつか別のコンパイラを使うはめになるかもしれないからです。

### *演習問題 2.2*

1）プログラム例 2.1 を入力し、コンパイルして実行してみなさい。

**解答例**　（省略）

2）プログラム例 2.1 を書き換えて、演習問題 2.1 の 3 ) に示した 3 つの式を計算するプログラムを作りなさい。このとき、計算式のデータ型を考慮して、変数の型を変更しなさい。結果を出力する時に、printf() 関数で用いる書式文字を間違えないように注意しなさい。演習問題 2.1 の 3 ) で考えた答えと、実際にプログラムで計算した結果を比較しなさい。

**解答例**　解答プログラム例を下記に示します。答え合ってますね。

**解答プログラム 2.2**　**簡単な計算のプログラム**

```
/*****************************************
    p2_2.c
    簡単な計算の例題プログラム
    Tadaaki Shimizu 2006.11.02
*****************************************/
#include <stdio.h>
main()
{
    int  answer;      // 計算の答え
    long longAnswer; // 計算の答え (long 版 )

    answer = 12 + 3 / 2;
    printf("12 + 3 / 2 = %d¥n", answer);
    answer = - 2 * 6 - 5;
    printf("- 2 * 6 - 5 = %d¥n", answer);
    longAnswer = 54000L * (12000L / 1000L);
    printf("54000L * (12000L / 1000L) = %ld¥n", longAnswer);
    exit(0);
}


[実行例]
% ./p2_2
12 + 3 / 2 = 13
- 2 * 6 - 5 = -17
54000L * (12000L / 1000L) = 648000
%
```

3）プログラム例 2.1 を書き換えて、変数 answer の代わりに、演習 2.1 の 4 ) に示した変数のうち文法的に間違っている変数名をわざと使って、簡単なプログラムを作りなさい。このプログラムをコンパイルしたときに出るエラー・メッセージを書きとめて、よく読みなさい。

**解答例**　下記に筆者の利用しているコンパイラ (**gcc** バージョン 3.3.1) でのコンパイル例を示します。先にも述べたように、**tel_$number** は変数名として使えてしまいました。その他は、エラーとなってコンパイルに失敗しています。

注目して欲しい点は、どれも文法違反の変数名を使ってしまったという間違いという意味では同じエラーなのに、エラー・メッセージはかなり違うという点です。それぞれ、変数名の間違い方に合わせたエラー･メッセージが出力されているのです。しかし、初心者ならずとも、もっと手短に「変数名がおかしいよ」と教えてくれないものかと思うかも知れませんね。

**1) answer を b) の tel_$number に変更した場合**

```
% cc -o p2_3a p2_3a.c
% ./p2_3a
3 + 5 = 8
%
```

**2) answer を d) の 1_output に変更した場合**

```
% cc -o p2_3b p2_3b.c
p2_3b.c:10:9: invalid suffix "_output" on integer constant
p2_3b.c: 関数 `main' 内：
p2_3b.c:10: error: 構文解析エラー before numeric constant
p2_3b.c:12:5: invalid suffix "_output" on integer constant
p2_3b.c:13:27: invalid suffix "_output" on integer constant
%
```

**3) answer を f) の double に変更した場合**

```
% cc -o p2_3c p2_3c.c
p2_3c.c: 関数 `main' 内：
p2_3c.c:10: 警告：useless keyword or type name in empty declaration
p2_3c.c:10: 警告：空の宣言です
p2_3c.c:12: error: 構文解析エラー before '=' token
p2_3c.c:13: error: 構文解析エラー before "double"
%
```

**4) answer を g) の number of data に変更した場合**

```
% cc -o p2_3d p2_3d.c
p2_3d.c: 関数 `main' 内：
p2_3d.c:10: error: 文法エラー before "of"
p2_3d.c:12: error: `number' undeclared (first use in this function)
p2_3d.c:12: error: (Each undeclared identifier is reported only once
p2_3d.c:12: error: for each function it appears in.)
p2_3d.c:13: error: 構文解析エラー before "of"
[tadaaki@eckert 06C]$
```

### *演習問題 2.3*

次に示すプログラム例2.2のうち、文法的に誤っている部分と、文法的には正しくても読みやすさの点からよくないと思われる部分をあげて理由を述べなさい。また、それらの部分を訂正した後に、入力し、コンパイルして実行しなさい。

**解答例** 1) 文法的に間違っている箇所

15 行目： **printf()** 関数の書式文字列の "( ダブルクォート ) が閉じていない。

```
printf("32 + 14 * 2 = %d\n",answer);
```

16 行目： 文の最後に **;**( セミコロン ) が欠けている。

```
l_answer = 234578 + 45L;
```

2) 読みやすさの点からよくない箇所

12 行目他：変数名 **l_answer** の先頭文字の **l**( エル ) は **I**( アイ ) や **1**( イチ ) と間違いやすい。このような文字は単独で用いるべきではない。例えば、**longAnswer** のような変数名にすれば間違いを防げる。

16 行目： 定数 **234578** も、**long** 型であることを明示するために、**234578L** としておくことが望ましい。

### *演習問題 2.4*

プログラムが文法的に正しくても、プログラムの動作に問題が生じる場合がある。例えば、short 型で、10000に 10000を2回乗算するようなプログラムを作成して、コンパイルし、実行してみなさい。結果はどのようになりましたか。

**解答例** 解答例を解答プログラム 2.4 に示します。実は、**short** 型では、**10000×10000** を行った時点で演算結果がおかしくなるので、解答プログラム 2.4 では 1 回目の乗算と 2 回目の乗算を両方計算して表示するようにしてあります。

まず、「文法的に正しいのに、プログラムの動作に問題が生じるのは何故？」と思われたかもしれません。これは、日本語のような普通の言語で考えるとすぐに納得がいきます。例えば、「コンピュータが私を食べた。」という文は、文法的には正しい日本語です。けれども、常識的には意味が通じない文です。これと同じで、C 言語の文法的に正しくても、思った通りに動作しない ( 意味の通じない ) プログラムはいくらでも作れてしまいます。しかも、C 言語のコンパイラは、文法誤りは見つけてくれますが、動作をおかしくする論理的な誤りはほとんど見つけてくれません。このため、論理的な誤りをデバッグすることは、非常に面倒な作業になってしまいます。

論理的な誤りには、大雑把に言って、1) プログラムとして書かれた手続き自体に何らかの考え違いがある場合と、2) 手続き自体は正しいが、コンピュータの物理的な制限によって上手く動かないという場合があります。この演習問題は、2 番目の場合です。**10000×10000** を計算した時点で、**short** 型が表せる値の範囲を超えてしまうのです。このような現象をオーバーフローと呼んでいます。演算結果がオーバーフローを起こすと、その値は正しい結果と食い違ってしまいます。

解答プログラム 2.4 は、かなりわざとらしい作り方をしています。例えば、

```
answer = 10000;
answer = answer * answer;
```

といった書き方です。

```
answer = 10000 * 10000;
```

と書いてもよさそうです。こちらでも動作はほぼ同じですが、オーバーフローが起ることをコンパイラがワーニングとして知らせてくれます。定数だけからなる計算はコンパイル時に行われるので、オーバーフローが起ることにコンパイラが気付くのです。一方で、たった 1 行加えて、乗算を変数からなる式に変えてやっただけで、コンパイラはオーバーフローが起ることに気付きません。

最後に、実行結果を見てみましょう。10000×10000 が負の数になるとは驚きですね。オーバーフローは、コンピュータが実際に行っている 2 進数演算のレベルでは、一定の規則に従って起ります。しかし、私たち人間から見れば、全くのデタラメと見えるような演算結果になってしまいます。もし、皆さんが作ったプログラムが得体の知れない演算結果を出力したら、第一の容疑者はオーバーフローのようなコンピュータの物理的な制限の中に潜んでいるかも知れません。

**解答プログラム 2.4**　**short 型のオーバーフロー**

```
/************************************
    p2_4.c
    short 型のオーバーフロー
    Tadaaki Shimizu  2006.11.06
************************************/
#include <stdio.h>

main()
{
    short answer;

    answer = 10000;
    answer = answer * answer;
    printf("10000 * 10000 = %hd¥n", answer);
    answer = 10000;
    answer = answer * answer * answer;
    printf("10000 * 10000 * 10000 = %hd¥n", answer);
    exit(0);
}
```

**[実行例]**

```
% ./p2_4
10000 * 10000 = -7936
10000 * 10000 * 10000 = 4096
%
```